Freescale Semiconductor
Application Note

Document Number: AN4020
Rev. 0, 01/2010

Calibrating the 16 MHz IRC on the MPC5510

by: Alasdair Robertson
Applications Engineering
Microcontroller Solutions Group. East Kilbride

1 Introduction

The Freescale Semiconductor MPC5510 family of microcontrollers has a 16 MHz Internal Reference Clock (IRC) that is used as the default system clock when the device is powered on. The IRC has an associated 8-bit trim field that is set during factory test to trim the frequency as close to 16 MHz as possible. The trim bits are stored into the flash shadow block and can then be copied into the IRC trim field by user code.

This application note describes a method that can be used to re-calibrate the 16 MHz IRC against an external crystal without the use of any external components such as signal generators or oscilloscopes. This is useful if the factory trim value in the shadow block is erased or corrupted, or if the trim value requires to be verified during different operating conditions.

For more information, refer to the MPC5510 Reference Manual and DataSheet at www.freescale.com/mpc55xx.

Contents

The MPC5510 family also has a 32 KHz Internal Reference Clock. All references to the IRC within this application note refer to the 16 MHz IRC unless otherwise explicitly stated.

2 Calibration Background

The 16 MHz IRC is not intended as an accurate reference clock and is subject to jitter and inaccuracy as defined in the MPC5510 DataSheet. Due to the manufacturing process, every device will exhibit a different IRC base frequency.

In order to compensate for these deviations, there is an 8-bit trim value in the SIU Clock Source Register (CLKSRC) that can be programmed to bring the IRC clock frequency as close to 16 MHz as possible.

3 Calibration Options

There are various methods that can be used to calibrate the IRC. These can be broken down to methods requiring external hardware and methods that can be run standalone.

3.1 External Hardware

If you have access to the CLKOUT pin (PE6) and have a calibrated oscilloscope, the following method can be used to trim the IRC.

- Ensure that the system is being clocked by the IRC (SIU SYSCLK register).
- Enable CLKOUT on PE6 and ensure the CLKOUT divider factor is set to 1 (SIU ECCR register).
- Monitor the frequency on PE6 and modify the IRC trim values in the CLKSRC register.

3.2 Standalone Trim

If you have no access to an oscilloscope, use the method outlined below to trim the IRC against an external crystal. This application note will focus on this method.

A typical 8 MHz or 16 MHz external crystal has a quoted accuracy of around 50 ppm. This translates as ± 50 seconds accuracy over a million seconds, or ± 4 seconds a day. This is more than accurate enough to calibrate the IRC.

The calibration mechanism allows you to compare the IRC frequency against the frequency of the external crystal by comparing the values of two counters (one clocked by the IRC and the other clocked by the external crystal) over a period of time. After the difference between the counters is calculated, the IRC can be adjusted via the trim bits to set the IRC as close as possible to 16 MHz.

NOTE

In this application note, the difference between two counters is referred as "TimeDelta".

4 The IRC Calibration Process

Figure 4-1 shows the steps involved in the calibration process. Each step is then explored in more detail using application code where necessary, to explain the full calibration methodology.

NOTE

The example provided in this application note assumes that an external 16 MHz crystal is used. In this instance, there is a 1 to 1 correlation between the counter clocked from the IRC and the counter clocked from the external crystal. By applying a suitable multiplier or divider to the value of the counter clocked by the external crystal, any permitted (within specification) crystal can be used.

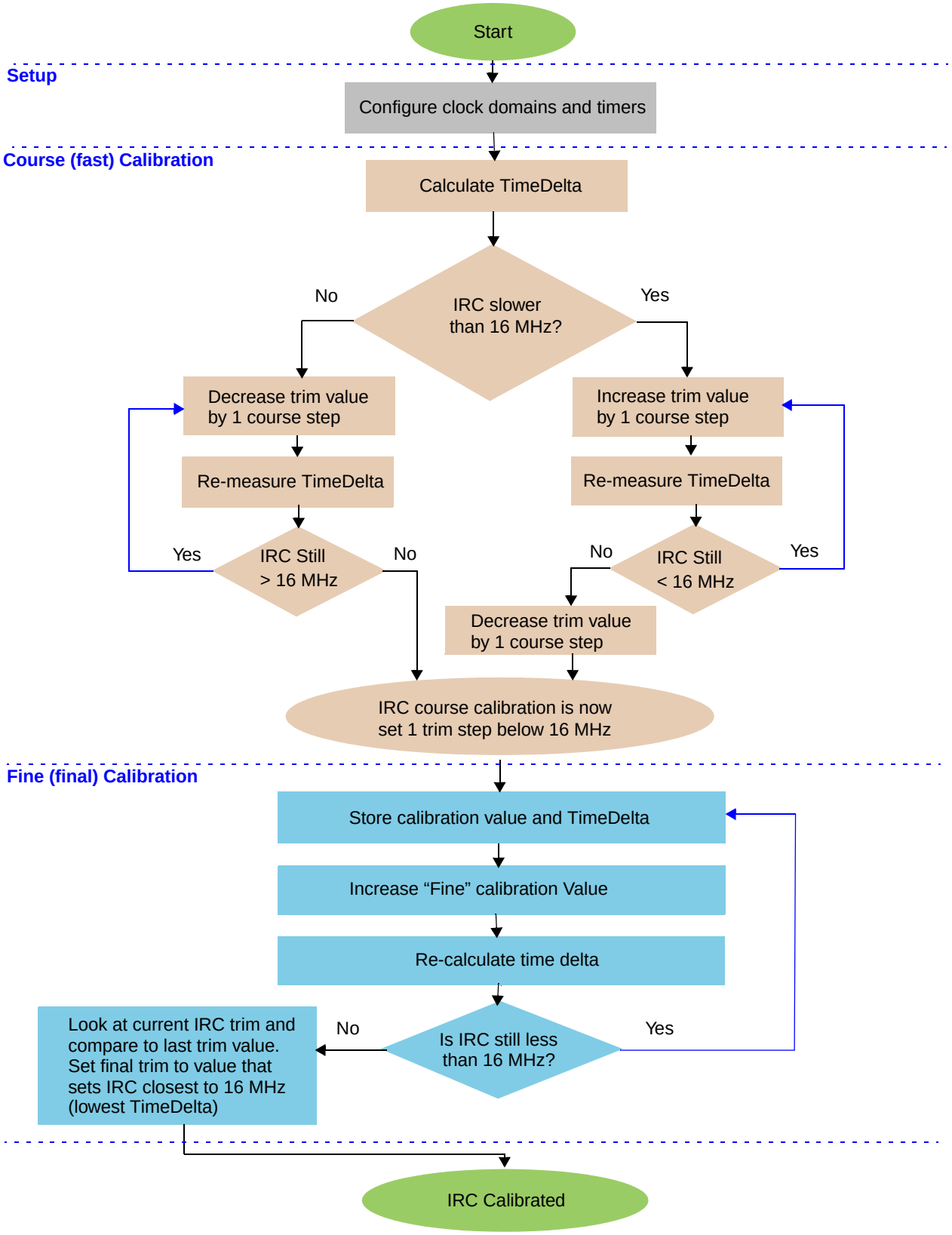

Figure 4-1. IRC calibration flow

4.1 Using the IRC Calibration Fields

Before looking at the details of calibration mechanism, it is important to understand the operation of the TRIMIRC calibration field in the SIU Clock Source Register (CLKSRC).

Offset: CRP_BASE + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	32KIRC EN	XOSC EN	0	32KOSC EN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0[1]	1	0	0[1]

	16	17	18	19	20	21	22	23	24[2]	25[2]	26[2]	27	28	29	30	31
R	TRIM32IRC[0:7]								TRIMIRC[0:7]							
W																
Reset	1[2]	1[2]	0	1	1	1	1	1	1	0	0	0	1	1	1	1

1 These bits are only reset by power-on, VDD15 LVI, VDD33 LVI, VDDSYN LVI, VDD5 Low LVI, and VDD5 LVI.
2 These bits must not be changed.
3 These bits must remain set to a value of 1. Only the six least significant bits of TRIM32IRC are used.

Figure 4-2. SIU Clock Source Register (CLKSRC)

If you have looked at the TRIMIRC field previously, you may have noticed that the order of the bits is unusual. The table below shows the bit order. In the CLKSRC register, the TRIMIRC bits are ordered 0 to 7, where bit-0 is the MSB (power architecture naming convention).

If we look at the actual active bit order in the TRIMIRC field, where bit 0 is defined as the bit that has the least effect on the trim, the actual active bit order of TRIMIRC is 2-1-0-7-6-5-4-3

Table 4-1. TRIMIRC active bit ordering

TRIMIRC Register Fields (0=MSB)	0	1	2	3	4	5	6	7
Corresponding active bit order (0 = Least effect on trim)	2	1	0	7	6	5	4	3
	Fine Trim			Course Trim				

This means that the upper 3 bits in the TRIMIRC field within the SIU CLKSRC register are the bits that have the least impact on the IRC trim. For clarity, these bits are called “Fine Trim" bits in this application note. The lower 5 bits in the TRIMIRC register have the most impact on trim and are therefore called the "Course Trim" bits.

NOTE

The MPC5510 Reference Manual states that only the 5 least significant bits of the TRIMIRC field should be used (see CLKSRC register description). The original intent was that these bits were used for a factory only trim factor, but this no longer applies. Therefore, all 8 TRIMIRC bits are available for user trim.

To illustrate the effect of the TRIMIRC bits, some tests were run on two MPC5517 RevA devices. The aim of this test is to show the effect on the IRC frequency when the fine and course trim bits are altered. Initially the IRC was trimmed to 16 MHz using the code in this application note. This acts as a centre point to look

at the effect of modifying the TRIMIRC bits by fine and course steps. IRC frequency figures are rounded to 2 decimal places.

CAUTION

The results shown below are solely to illustrate the effect of the IRC trim bits. The data recorded in the table is only valid for that specific device and cannot be applied as calibration data for any other device. The IRC characteristics vary significantly between devices requiring each device to have it's IRC individually calibrated.

Table 4-2. Effect on TRIMIRC bits on IRC frequency

Part	TRIMIRC	IRC Frequency	Comments
PPC5517GMLU66 XEAA0816 (176 QFP)	0x4F	16.27 MHz	1 Course increase step
	0x8E	16.07 MHz	2 fine increase steps
	0x6E	16.03 MHz	1 fine increase step
	0x4E	15.99 MHz	Trim value after IRC algorithm run (Centre point)
	0x2E	15.96 MHz	1 fine decrease step
	0x0E	15.93 MHz	2 fine decrease steps
	0x4D	15.68 MHz	1 course decrease step
PPC5517GMMG66 CTEQG0815 (208 BGA)	0x4C	16.38 MHz	1 Course increase step
	0x8B	16.07 MHz	2 fine increase steps
	0x6B	16.03 MHz	1 fine increase step
	0x4B	16.00 MHz	Trim value after IRC algorithm run (Centre point)
	0x2B	15.96 MHz	1 fine decrease step
	0x0B	15.93 MHz	2 fine decrease steps
	0x4A	15.67 MHz	1 course decrease step

From the very limited sample size of 2 parts, it can be seen that a fine trim step (± 0x20 to TRIMIRC, assuming none of the course trim bits are affected) changes the frequency by approximately 0.03-0.04 MHz. A course trim step (± 0x01 to TRIMIRC) changes the frequency by approximately 0.3 MHz.

4.2 Calculating the Delta Between Clocks

As described above, the calibration method used to calculate delta between the IRC and external crystal is relatively simple by using two counters, each clocked by the different clock source. By looking at the difference between the counters (TimeDelta) over a set period, the relative frequency difference can be calculated.

IRC clocked counter — The Real Time Counter (RTC) is a free running 32-bit counter that can be clocked from the 32 KHz IRC or from the 16 MHz IRC (with or without a divide by 512 prescaler). In this application note, the RTC will be clocked directly from the IRC. There is also a fixed divide by 32 prescaler on the RTC from the chosen clock, so this has to be factored into the results calculations. When the RTC is enabled, it starts counting up from 0x0. The RTC can also trigger an interrupt after a certain user defined duration, but this is not required in this example.

External crystal clocked counter — The Periodic Interrupt Timer (PIT) module has a dedicated Real Time Interrupt (RTI) timer that is always clocked directly from the external oscillator with no prescaler. Unlike the RTC, the PIT RTI is a decrementing counter meaning, it must be pre-loaded with a counter value and will then decrement from this value to 0x0. Once the counter reaches 0x0, a flag is set and the counter is automatically re-loaded with the pre-loaded value, and the process starts again.

To measure the difference between the IRC and external crystal, the system clock is set to run from the 16 MHz IRC (default setting). The RTC is also set to be clocked directly from the 16 MHz IRC. The PIT RTI is pre-loaded with a value that corresponds to the time that the counters will be compared over (in this example, 0.5 seconds, which is 8,000,000 counts). Both counters are then started and the application software monitors the RTI flag (to indicate the RTI counter has decremented to 0x0). As soon as the RTI flag is detected, the RTC can then be read. A direct comparison is made between the 2 counter values (after applying the ×32 multiplier on the RTC clock) to calculate the TimeDelta. The aim of the IRC calibration software is to get this TimeDelta as small as possible, therefore setting the IRC frequency as close as possible to 16 MHz.

The diagram below shows the configuration and measurement steps on a timeline chart. The blue solid line is the measurement phase. To maintain accuracy, the RTC must be read as soon as the PIT RTI flag is detected.

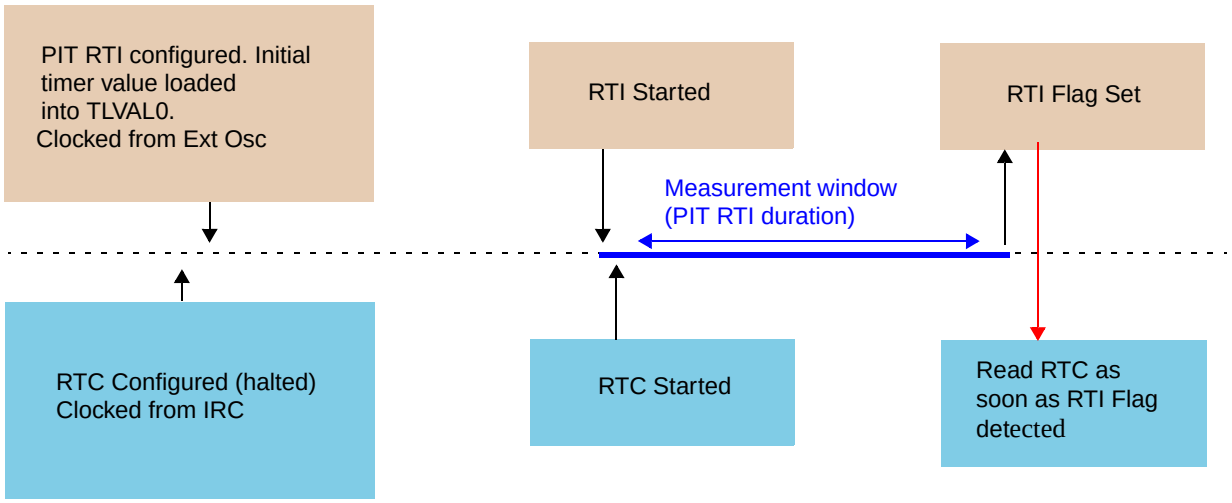

Figure 4-3. TimeDelta measurement

There are some timing errors with this approach as listed below.

- It is not possible to start the counters concurrently. To minimize this, the PIT RTI is started first and then the RTC is started. At the end of the cycle, the RTI flag is polled in software until it is set and then the RTC counter is read immediately afterwards. There is a varying error in the time between

the flag being set and the software detecting it depending where the program counter is sitting in the while loop.

- There are two counters running in separate asynchronous clock domains. There is a fixed delay between 4 and 9 clocks when reading across domains (for the PIT RTI counter being accessed from the RTC clock domain).
- There is an additional margin of error when reading two counter values (up to just under ± one clock per counter read).

In this example, these errors are insignificant when measuring over an 8 million clock cycle period. The period could safely be significantly reduced without any impact to the overall calibration accuracy. In the case of the application example, the period is set to 0.5 seconds to allow the IRC trim steps to be seen on an oscilloscope if desired.

4.3 Setup

NOTE

It is assumed that the basic MPC5510 start-up tasks (initializing the RAM, setting up the stack, and configuring the MMU) have already been carried out by standard initialization code and are not covered here.

The calibration setup includes:

1. Disabling the watchdog. (Not calibration specific, but often forgotten!)
2. Ensuring the system is clocked from the 16 MHz IRC.
3. Setting an initial starting "guess" value for the IRC trim. This saves a bit of time in the calibration routine. A safe guess is 0x0D.
4. Setting up the RTC and PIT RTI counters.

Sample Code:

```
MCM.SWTCR.B.SWE = 0;                /* Disable watchdog                                   */
SIU.SYSCLK.R = 0x0;                 /* Ensure system clocked by IRC (default)             */

CRP.CLKSRC.B.TRIMIRC = 0x0D;        /* set "good guess" trim value                        */

/* Setup RTC */
CRP.RTCSC.R = 0x0;                  /* Clear RTC to default state (disables and resets counter) */
CRP.RTCSC.B.CLKSEL = 0x3;           /* Set RTC clock source to be direct 16 MHz IRC       */

/* Setup PIT RTI */
PIT.EN.B.PEN0=0;                    /* Disable PIT RTI to clear counter                   */
PIT.TLVAL0.R = 0x007A1200;          /* Load 0.5 second tick to TLVAL0                     */

/* Optional Enable CLKOUT. Not required but allows you to see the trim taking place if desired */
SIU.PCR[70].R = 0x060C;             /* Setup CLKOUT on PE6                                */
SIU.ECCR.B.EBDF = 0x0;              /* CLKOUT divider =1 (mirrors system clock, i.e. IRC) */
```

Figure 4-4. Setup code

4.4 Time Difference Calculation

The theory behind calculating the time difference is described in Section 4.2. The sample code to implement this is shown below. Typically, this is implemented as part of a function because the same code is called many times throughout the calibration process.

Variables Used:

- **RTCread1** — Local unsigned 32-bit integer containing the RTC counter value.
- **TimeDelta** — Global signed 32-bit integer containing the time delta between counters. Note the last line of code where this is actually calculated. 0x007A1200 corresponds to 8 million in decimal (0.5s) and is the period of the RTI counter that was pre-loaded into the RTI during setup.

Sample Code:

```
void clockdiff(void)
{
 uint32_t RTCread1;                    /* Local Variables, Unsigned 32-bit Integer                    */

 /* Clear counters */
  CRP.RTCSC.B.CNTEN = 0x0;             /* Stop the RTC to clear the counter                           */
  PIT.EN.B.PEN0=0;                     /* Disable PIT RTI which stops and re-loads RTI counter        */
  PIT.FLG.B.RTIF = 0x1;                /* Clear the RTIF flag so that it can be set when counter=0    */

  /* Start the counters running */
  PIT.EN.B.PEN0=1;                     /* Start the PIT RTI first (count down from 0x007A1200)        */
  CRP.RTCSC.B.CNTEN = 0x1;             /* Then start the RTC (free running count up from 0)           */

  /* Wait for PIT RTI to set a flag once counter has decremented from 0x007A1200 to 0x0 (0.5 seconds)  */
  while (PIT.FLG.B.RTIF == 0);
  RTCread1 = CRP.RTCCNT.R;             /* and then immediately read RTC counter value                 */

  /* Now calculate the time difference between RTC (IRC) and PIT RTI (Ext Clock) */
  RTCread1 = RTCread1 * 32;            /* RTC has fixed divide by 32 prescaler                        */
  TimeDelta = (0x007A1200 – RTCread1); /* Calculate difference (+ve or –ve) between the counters      */

} /* --- End of ClockDiff --- */
```

Figure 4-5. Counter difference (TimeDelta) calculation code

4.5 Course (Fast) Calibration

As mentioned in Section 4.1, the TRIMIRC field in the CLKSRC register has been effectively split into a fine trim section (upper 3 bits) and a course trim section (lower 5 bits).

The first part of the course calibration process is to run the counter calibration routine to obtain the TimeDelta value. The TimeDelta value defines if the IRC is too fast or too slow.

Due to the calculation of the TimeDelta (RTI counter – RTC counter), if the delta is +ve, then the external clock (clocking RTI) is running faster than the IRC (clocking the RTC), that is the IRC is running too slow.

The calibration routine is split into two parts, one for the case where the IRC is slower than 16 MHz and other where it is faster 16 MHz.

Course calibration function:

- Get current TimeDelta value.
- If IRC is slower than 16 MHz:
  a) Increment the IRC trim value by 1 "course" step (add 1 to TRIMIRC field).
  b) Get new TimeDelta value.
  c) Repeat until IRC is greater than 16 MHz; at that point, subtract 1 from TRIMIRC field.
- If IRC is faster than 16 MHz:
  a) Decrement the IRC trim value by 1 "course" step (subtract 1 from TRIMIRC field).
  b) Get new TimeDelta value.
  c) Repeat until IRC is less than 16 MHz.

In the example code, there is a global variable "IRCTrimVal" that holds the current TRIMIRC value.

One of the important features of the course calibration routine is that it always exits with the TRIMIRC value set such that the IRC is trimmed to one course step less than 16 MHz. This makes the subsequent fine trimming algorithm a lot simpler and faster.

NOTE

This is called the "fast" calibration phase as the IRC trim values are incremented by course amounts so each step results in a faster jump towards 16 MHz. This can be seen by looking at the CLKOUT on an oscilloscope where the IRC frequency initially jumps in large steps as the course calibration routine runs.

Sample Code:

```
void FastTrim (void)                    /* Course trim of IRC using trim bits 3..7                    */
{
 uint8_t activetrim=1;                  /* Local Variable, Unsigned 8-bit Integer                     */
 clockdiff();                           /* 1st thing to do is to get current "TimeDelta"              */
 if (TimeDelta > 0)                     /* IRC is slower than 16 MHz crystal                          */
 {
  while (activetrim==1)                 /* activetrim=1 means still trimming. Set to 0 to stop trim   */
  {
   IRCTrimVal =IRCTrimVal+1;                    /* Add 1 course step to IRC trim value               */
   CRP.CLKSRC.B.TRIMIRC = IRCTrimVal;           /* and write it to trim register (IRC changes freq)  */
   clockdiff();                                 /* re-measure delta with new trim value              */
   if (TimeDelta < 0)                   /* Check if IRC is faster than external clock (over trimmed)  */
   {
     activetrim=0;                              /* If so, then remove 1 step from trim               */
     IRCTrimVal =IRCTrimVal-1;                  /* by simply deducting 1 from trim value             */
     CRP.CLKSRC.B.TRIMIRC = IRCTrimVal;         /* and updating trim register                        */
     clockdiff();                               /* Update TrimDelta field again                      */
   }
  } /* end of while activetrim */
} /* End of TimeDelta > 0 */

if (TimeDelta < 0)                      /* IRC is faster than 16MHz crystal                           */
 {
  while (activetrim==1)                 /* activetrim=1 means still trimming. Set to 0 to stop trim   */
  {
    IRCTrimVal =IRCTrimVal-1;                   /* Subtract 1 course step from IRC trim value        */
    CRP.CLKSRC.B.TRIMIRC = IRCTrimVal;          /* and write it to trim register (IRC changes freq)  */
    clockdiff();                                /* re-measure delta with new trim value              */
    if (TimeDelta > 0)  activetrim=0;   /* If IRC is now slower that 16MHz crystal then complete      */
  } /* end of while activetrim */
 } /* End of TimeDelta < 0 */
} /* End of FastTRim */
```

Figure 4-6. Course trim function

4.6 Fine Calibration

The fine TRIMIRC adjustment is the final phase in the IRC calibration process. Entry into this function assumes that the IRC has already been trimmed using the course TRIMIRC bits and is trimmed such that the IRC is less than 16 MHz.

The first step is to measure and store the current TimeDelta along with the current TRIMIRC value. The fine calibration algorithm will then increase the TRIMIRC value until the IRC frequency is just above 16 MHz, storing the previous TimeDelta and TRIMIRC values each time. Once the IRC frequency is higher than 16 MHz, the adjustment is stopped and the current and previous TimeDelta values are compared. The smallest delta is selected and the TRIMIRC is set accordingly, therefore giving the closest frequency to 16 MHz.

NOTE

During the adjustment phase, if the TRIMIRC fine bits have all been set (TRIMIRC = 0b111xxxxx), the next course bit is set and fine bits are cleared. At this point, the IRC frequency will be greater than 16 MHz as the course algorithm leaves the IRC at one course trim step less than 16 MHz.\

Fine Calibration Function:

1. Get current TimeDelta value.
2. Store current TimeDelta and TRIMIRC value.
3. Increment TRIMIRC by 1 fine step:
   - If exceeded the limit of fine trim (TRIMIRC = 0b111xxxx), then set fine trim bits to zero and increment next course trim. By definition, this should make IRC frequency more than 16 MHz. Apply new TRIMIRC value.
   - Otherwise, add 0x20 (1 normal fine trim step). Apply new TRIMIRC value.
4. Get new TimeDelta value.
5. If IRC is faster than 16 MHz:
   - Compare current (new) TimeDelta with previous value and work out which is better. and then program the current or previous TRIMIRC value to get optimum frequency. Write trim value and time delta to RAM.

   Else:
   - Jump back to Step 2.

```
void FineTrim (void)                    /* Fine trim of IRC using trim bits [0..2]                    */
{
  uint8_t activetrim=1;                 /* Local Variable, Unsigned 8-bit Integer                     */
  uint8_t LastTrim;                     /*    Used to store previous setting of TRIMIRC                */
  int32_t LastDelta;                    /*    Used to store previous value of TimeDelta               */

  clockdiff();                          /* get current TimeDelta                                      */

  while (activetrim==1)                 /* activetrim=1 means still trimming. Set to 0 to stop trim    */
  {
    LastDelta = TimeDelta;              /* Store previous TimeDelta                                   */
    LastTrim  = IRCTrimVal;             /*     and previous TRIMIRC value                             */

    /* Increment trim value by 1 step                                                                 */
    if (IRCTrimVal > 0xE0)              /* Exceeded limit of fine trim (TRIMIRC=0b111xxxxx)            */
      {                                 /*  so set fine trim back to 0 and increment course trim      */
        IRCTrimVal = IRCTrimVal - 0xE0 + 0x1;   /* by 1 count. Should result in IRC > 16 MHz            */
      }
    Else  IRCTrimVal =IRCTrimVal + 0x20;        /* Otherwise increment fine trim by normal step       */

    CRP.CLKSRC.B.TRIMIRC = IRCTrimVal;  /* Apply new trim value (IRC changes)                         */
    clockdiff();                        /* and re-calculate the TimeDelta                             */

    if (TimeDelta < 0)      /* Keep increasing trim until IRC is faster than 16MHz (TineDelta<0)      */
    {
      if (LastDelta < 0) LastDelta = (0-LastDelta);     /* if LastDelta is –ve, make it +ve           */
      if (TimeDelta < 0) TimeDelta = (0-TimeDelta);     /* and same for TimeDelta                     */

      if (LastDelta < TimeDelta)    /* If previous TRIMIRC setting gave better TimeDelta then         */
        {                           /*  revert to previous TRIMIRC (saved in LastTrim)                */
          IRCTrimVal = LastTrim;
          CRP.CLKSRC.B.TRIMIRC = IRCTrimVal;    /* Apply previous TRIMIRC, IRC changes                  */
          clockdiff();                          /* and re-read TimeDelta                                */
        }

      activetrim=0;             /* Signal that  no more trimming is required. Got optimal value       */

    } /* end of "if (TimeDelta < 0) */
  }  /* end of while activetrim */
} /* End of FineTRim */
```

Figure 4-7. Fine trim function

5 Summary

This application note has outlined a method that can be used to trim the 16 MHz IRC on the MPC5510.

This could be further expanded to automatically program this value into the shadow block of the flash, but this is application code dependent (depends what is in the shadow block). The only generic "safe" solution would be to copy the contents of the shadow flash to RAM, modify the TRIMIRC byte, erase the shadow flash, and then program the contents back in. This is however out with the scope of this application note.

Appendix A main.c

```
/*****************************************************************************/
/* MPC5510 16Mhz IRC Trim Software                                           */
/* Alasdair Robertson, Freescale EKB Applications                            */
/*                                                                           */
/* v0.1  15Nov09   AR     Initial Version                                    */
/*                                                                           */
/* Compares IRC to external 16Mhz crystal and provides closest IRC trim      */
/*  value. Value is returned in RAM at address 0x4000_0020. This can         */
/*  then be programmed into shadow row. This code does not automatically     */
/*  program trim value to shadow flash. That is up to the user               */
/*                                                                           */
/* IRC can be trimmed to approx 0.1Mhz but the accuracy obviously            */
/*  depends on the accuracy of the 16Mhz crystal!                            */
/
/*                                                                           */
/*  Set breakpoint on while(1) in main to indicate trim value achieved       */
/*                                                                           */
/*****************************************************************************/
#include "..\header\mpc5516.h"
#include "..\header\typedefs.h"

#define CLKOUT_ON;

#define PIT_INIT_VAL 0x007A1200;     /* Compare Window for algorithm    */
#define IRC16TRIM  0x0D;

/* Global Variables */
int32_t TimeDelta;                  /* Diff between IRC and Ext Clock   */
uint8_t IRCTrimVal;                 /* Current Trim Value               */

void clockdiff(void);
void FastTrim (void);
void FineTrim (void);

void main()
{

  MCM.SWTCR.B.SWE = 0;       /* Disable watchdog                        */
  SIU.SYSCLK.R = 0x0;        /* Ensure system clocked by IRC (default)  */

  IRCTrimVal = IRC16TRIM;              /* Read "good guess" trim value  */
  CRP.CLKSRC.B.TRIMIRC = IRCTrimVal;   /*  and program to IRC Trim      */

  /* Trim routine works by using PIT RTI (Clocked by External OSC)      */
  /*  to measure 0.5 second. RTC (clocked by IRC) is then compared to   */
  /*  PIT timer for same duration and delta calculated                  */

  /* Setup RTC */
  CRP.RTCSC.R = 0x0;                /* ensure RTC is default state      */
  CRP.RTCSC.B.CLKSEL = 0x3;         /* RTC clocked direct from 16MhzIRC */
```

```
  /* Setup PIT */
  PIT.EN.B.PEN0=0;                  /* Disable PIT RTI to clear counter */
  PIT.TLVAL0.R = PIT_INIT_VAL;      /* Load 0.5 second tick to TVAL0    */

  #ifdef CLKOUT_ON
    SIU.PCR[70].R = 0x060C;         /* Setup CLKOUT on PE6              */
    SIU.ECCR.B.EBDF = 0x0;          /*  and divider to 1                */
  #endif

  FastTrim();           /* Initial course trim (using trim bits 3..7)   */

  FineTrim();           /* Fine trim using trim bits 0..2               */

  /* Write out trim value to RAM at 0x4000_0020 */
  (*(vuint32_t *)0x40000020) = 0x0;
  (*(vuint8_t *)0x40000020) = IRCTrimVal;

  /* Write out Delta to 0x4000_0024 */
  (*(vuint32_t *)0x40000024) = TimeDelta;

  /* Set breakpoint here to indicate completion */
  while(1);

} /* --- END of Main --- */

void clockdiff(void)
/* Calculate the delta between External 16Mhz Osc and 16Mhz IRC         */
{

  uint32_t RTCread0, RTCread1, RTCread2, PITread1, PitInit;
  vuint32_t dummy;

  /* Clear counters */
  CRP.RTCSC.B.CNTEN = 0x0;          /* Stop and clear RTC                */
  PIT.EN.B.PEN0=0;                  /* Disable PIT RTI to reload counter */
  PIT.FLG.B.RTIF = 0x1;             /* Clear RTIF                        */

  /* Start counters running */
  PIT.EN.B.PEN0=1;                  /* Start PIT clocked by 16Mhz Osc    */
  CRP.RTCSC.B.CNTEN = 0x1;          /* Start RTC running (16Mhz IRC)     */

  while (PIT.FLG.B.RTIF == 0);      /* Wait for PIT to overflow (flag)   */

  RTCread1 = CRP.RTCCNT.R;          /*  and read RTC at that point       */

  /* Now calculate the time delta between IRC and Crystal */
  RTCread2 = RTCread1 * 32;         /* RTC has fixed /32 prescaler!      */
  PitInit = PIT_INIT_VAL
  TimeDelta = (PitInit - RTCread2);

}  /* --- End of ClockDiff --- */

void FastTrim (void)
/* Course trim of IRC using trim bits 3..7) */
{
  uint8_t activetrim=1;
```

```
  clockdiff();         /* get TimeDelta */

  if (TimeDelta > 0)
  {
  /* IRC is slower than external 16Mhz crystal                          */
  /* Speed up by incrementing TRIMIRC value. Always exit from this with  */
  /* IRC slower than external crystal so can then use fine adjust to     */
  /* Increase clock speed.                                               */

  while (activetrim==1)
   {

    IRCTrimVal =IRCTrimVal+1;
    CRP.CLKSRC.B.TRIMIRC = IRCTrimVal;
    clockdiff();  /* re-measure with new trim value */

    if (TimeDelta < 0)   /* Check if IRC is faster than external clock   */
    {
      activetrim=0;                /* If so, go to previous trim setting */
      IRCTrimVal =IRCTrimVal-1;
      CRP.CLKSRC.B.TRIMIRC = IRCTrimVal;
      clockdiff();
    }

   }  /* end of while activetrim */

    /* Note that this routine always exits with TimeDelta >= 0 so will   */
    /*  not execute the next "if" statement. Has to be coded in this     */
    /*  order however!                                                   */
    /* ElseIf may not work since dependent variable has been changed     */

  } /* End of TimeDelta > 0 */

 if (TimeDelta < 0)
 {
  /* IRC is faster than external 16Mhz crystal                          */
  /* Slow down by decrementing TRIMIRC value. Always exit from this with */
  /* IRC slower than external crystal so can then use fine adjust to     */
  /* Increase clock speed.                                               */

  while (activetrim==1)
   {
    IRCTrimVal =IRCTrimVal-1;
    CRP.CLKSRC.B.TRIMIRC = IRCTrimVal;
    clockdiff();  /* re-measure with new trim value */

    if (TimeDelta > 0) /* Complete. IRC is now slower than Ext Clock)    */
     {
      activetrim=0;
     }

   }  /* end of while activetrim */

  }  /* End of TimeDelta < 0 */

} /* End of FastTRim */
```

main.c

```
void FineTrim (void)
/* Fast trim leaves IRC trimmed such that it's slower than external     */
/*   16Mhz. Now trim "fine" bits [0..2] to get IRC that is as close as  */
/*   possible to external                                               */
{

  uint8_t activetrim=1;
  uint8_t LastTrim;
  int32_t LastDelta;

  clockdiff();             /* get current TimeDelta                    */

  while (activetrim==1)
   {

    LastDelta = TimeDelta;   /* Store previous delta to use as compare  */
    LastTrim  = IRCTrimVal;  /* and store previous trim value           */

    /* Increment Trim Value. If current "fine" adjust (ie bits 0..2)    */
    /*  are at their limit, then increment course adjust and clear fine */
    if (IRCTrimVal > 0xE0)
     {
        IRCTrimVal = IRCTrimVal - 0xE0 + 0x1;   /* Fine to 0, course +1 */
     }
    else
     {
         IRCTrimVal =IRCTrimVal + 0x20;         /* Normal increment fine */
     }

    CRP.CLKSRC.B.TRIMIRC = IRCTrimVal;         /* Write new trim value  */
    clockdiff();                       /* re-measure with new trim value */

    /* Keep trimming up to the point where IRC is faster than External  */
    /*  osc. At that point, compare the current and previous deltas to  */
    /*  see which gives the closest frequency                           */
    if (TimeDelta < 0)
    {

      /* Perform Modulus function on Deltas so can compare them         */
      if (LastDelta < 0) LastDelta = (0-LastDelta);
      if (TimeDelta < 0) TimeDelta = (0-TimeDelta);

      /* If previous settings were better than current, revert to them  */
      if (LastDelta < TimeDelta)
       {
         IRCTrimVal = LastTrim;
         CRP.CLKSRC.B.TRIMIRC = IRCTrimVal;
         clockdiff();
       }

       activetrim=0;

     } /* end of "if (TimeDelta < 0) */

   }  /* end of while activetrim */

} /* End of FineTRim */
```

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN4020
Rev. 0
01/2010

freescale™
semiconductor